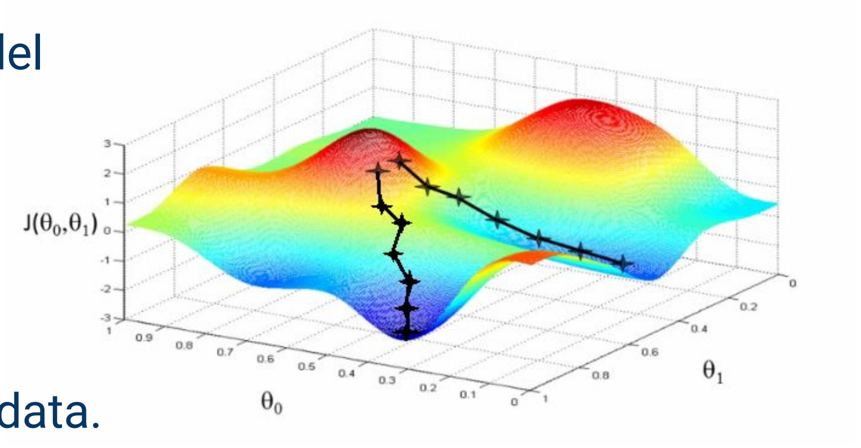
Gradient Descent

Gradient Descent

Training: the process of adjusting the model parameters to fit the given data.

Gradient descent: Repeatedly update parameters by taking small steps in the negative direction of the partial derivative, so the model gets better at predicting our data.



$$W = W - \alpha \frac{\partial L}{\partial W}$$

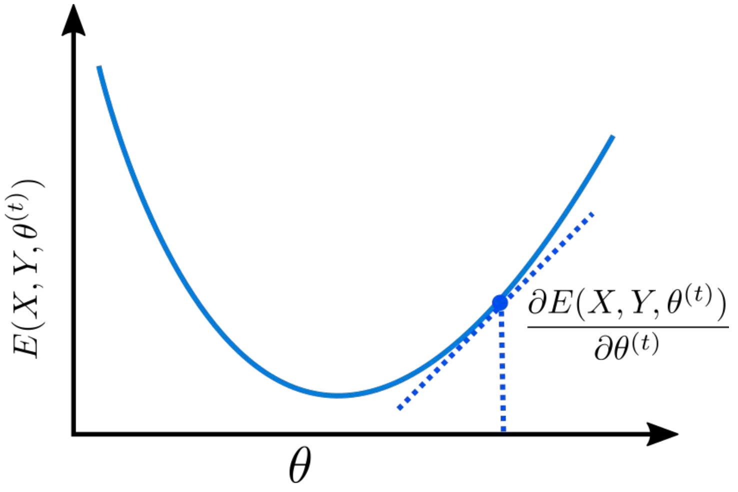
where α is the learning rate / step size.

This assumes that the gradients can be computed - all our network functions and the loss need to be differentiable.

Gradient Descent

Algorithm

1. Initialize weights randomly
2. Loop until convergence:
3. Compute gradient based on the **whole dataset**:
4. Update weights
5. Finish



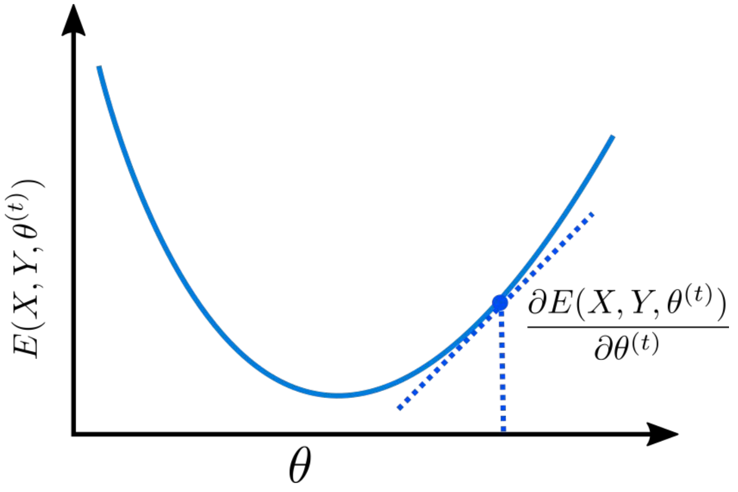
Tip: Compute all gradients before updating the weights!

In practice, datasets are often too big for this.

Stochastic Gradient Descent

Algorithm

```
1. Initialize weights randomly
2. Loop until convergence:
3.     Loop over each datapoint:
4.         Compute gradient based
           on the datapoint
5.         Update weights
6. Finish
```



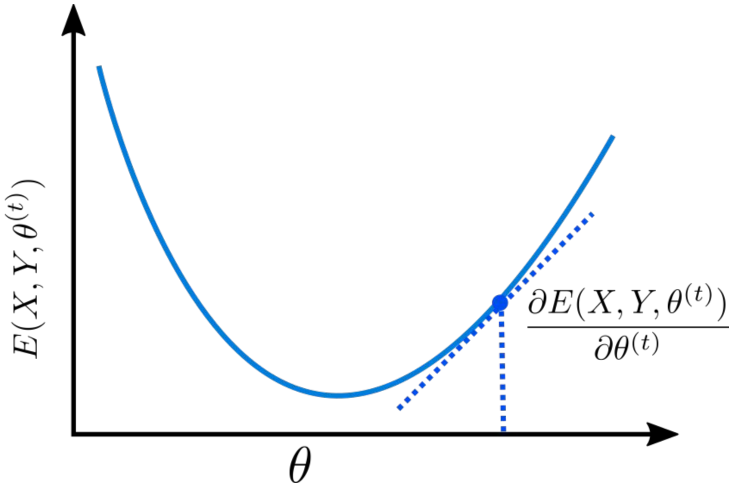
$E(X, Y, \theta^{(t)})$
$\frac{\partial E(X, Y, \theta^{(t)})}{\partial \theta^{(t)}}$
θ

Very noisy to take steps based only on a single datapoint

Mini-batched Gradient Descent

Algorithm

```
1. Initialize weights randomly
2. Loop until convergence:
3.         Loop over batches of datapoints:
4.                 Compute gradient based
                   on the batch
5.                 Update weights
6. Finish
```

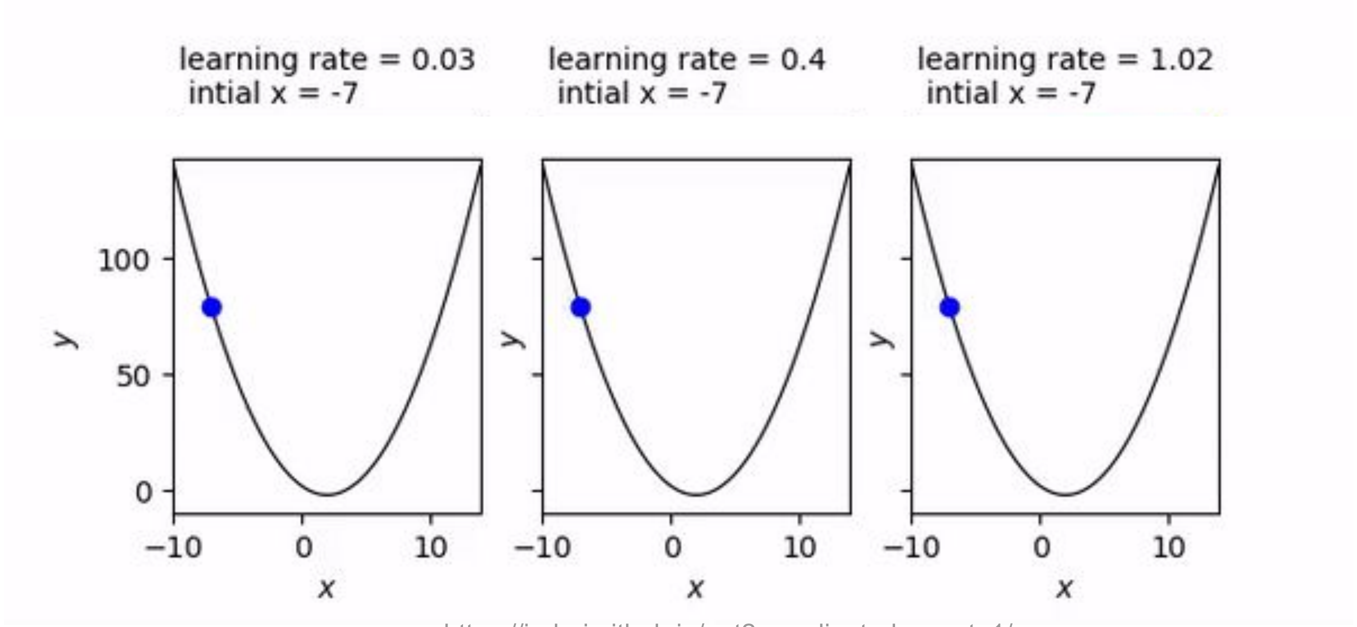

$E(X, Y, \theta^{(t)})$
$\frac{\partial E(X, Y, \theta^{(t)})}{\partial \theta^{(t)}}$
θ

This is what we mostly use in practice

The Importance of the Learning Rate

If the learning rate is too low, the model will take forever to converge.

If the learning rate is too high, we will just keep stepping over the optimal values.



https://jed-ai.github.io/opt2_gradient_descent_1/

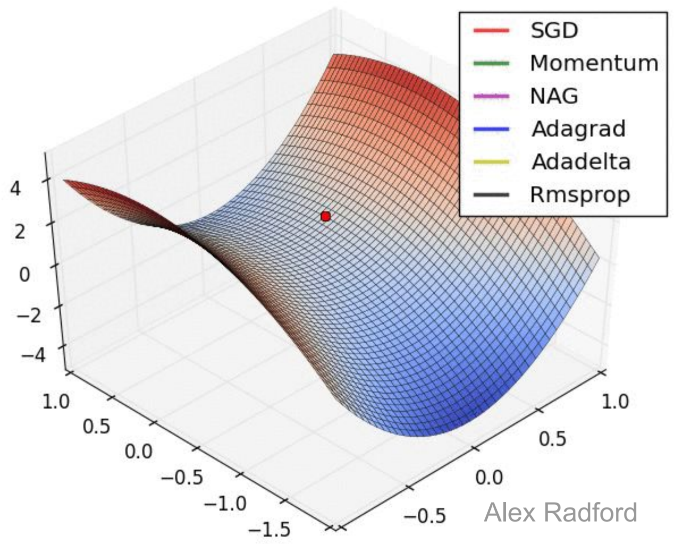
Adaptive Learning Rates

Intuition:

Have a different learning rate for each parameter.

Take bigger steps if a parameter has not been updated much recently.

Take smaller steps if a parameter has been getting many big updates.



Alex Radford

Tip: "Adam" and "AdaDelta" work quite well.

Learning rate decay

Reducing the learning rate by a factor:

$$\alpha \leftarrow \alpha d \qquad d \in [0, 1]$$

The intuition is to take smaller steps as we get closer to the minimum, so we don't overshoot it.

Strategies for performing the update:

- Every epoch
- After a certain number of epochs
- When performance on the validation set hasn't improved for several epochs.

Weight initialisation

What values do we set the weights to before training?

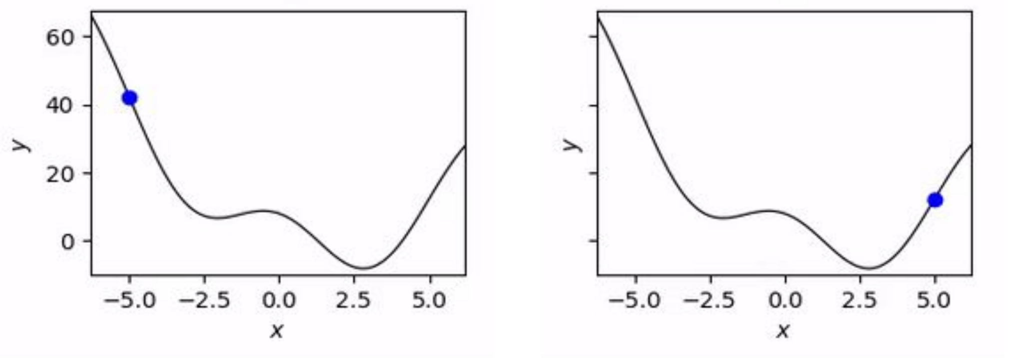
- **Zeros**: It is common to set the bias weights to zero, because we don't want the neurons to start out with a bias.
- **Normal**: Draw randomly from a normal distribution. Mean 0 and variance 1 or 0.1 are common choices.
  We want the weights to be different, or they all get the same updates!
- **Xavier Glorot** (named after one of the authors):

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

  where U is the uniform distribution, n_j the number of neurons in the previous layer, n_{j+1} the number of neurons in the next layer.

Randomness in the Network

Different **random initializations** lead to different results.



Solution: Explicitly set the random seed. All the random seeds!

BUT!

GPU threads finish in a random order, also leading to randomness! Small rounding errors really add up! Doesn’t affect all operations.

Solution: Embrace randomness, run with different random seeds and report the average.

Data normalisation

Min-max normalisation: Scaling the smallest value to *a* and largest value to *b*. For example [0, 1] or [-1, 1].

$$X' = a + \frac{(X - X_{min})(b - a)}{X_{max} - X_{min}}$$

Standardization (z-normalization): Scaling the data to have mean 0 and standard deviation 1.

$$X' = \frac{X - \mu}{\sigma}$$

Normalisation helps because weight updates are **proportional to the input**.

$$\frac{\partial Loss}{\partial W} = X^T \frac{\partial Loss}{\partial Z}$$

The scaling values need to be calculated based on the **training set only**!

Gradient checking

A way to check that your gradient descent is implemented correctly. Two alternative ways to isolate the gradient.

Method 1: from the weight difference before and after gradient descent

$$w^{(t)} = w^{(t-1)} - \alpha \cdot \frac{\partial L(w)}{\partial w} \qquad \frac{\partial L(w)}{\partial w} = \frac{w^{(t-1)} - w^{(t)}}{\alpha}$$

Method 2: actually changing the weight a bit and measuring the change in loss

$$\frac{\partial L(w)}{\partial w} = \lim_{\epsilon \to 0} \frac{L(w+\epsilon) - L(w-\epsilon)}{2\epsilon} \qquad \frac{\partial L(w)}{\partial w} \approx \frac{L(w+\epsilon) - L(w-\epsilon)}{2\epsilon}$$

Both methods should give very similar values of $\frac{\partial L(w)}{\partial w}$